```
 1  // joi/5/bank/Bank.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  import java.util.*;
 7
 8  /**
 9   * A Bank object simulates the behavior of a simple bank/ATM.
10   * It contains a Terminal object and a collection of
11   * BankAccount objects.
12   *
13   * The visit method opens this Bank for business,
14   * prompting the customer for input.
15   *
16   * To create a Bank and open it for business issue the command
17   * <code>java Bank</code>.
18   *
19   * @see BankAccount
20   * @version 5
21   */
22
23  public class Bank
24  {
25      private String bankName;           // the name of this Bank
26      private Terminal atm;              // for talking with the customer
27      private int balance = 0;           // total cash on hand
28      private int transactionCount = 0;  // number of Bank transactions
29      private Month month;               // the current month.
30
31      private TreeMap accountList;      // mapping names to accounts.
32
33      // what the banker can ask of the bank
34
35      private static final String BANKER_COMMANDS =
36      "Banker commands: " +
37          "exit, open, customer, report, help.";
38
39      // what the customer can ask of the bank
40
41      private static final String CUSTOMER_TRANSACTIONS =
42      "    Customer transactions: " +
43      "deposit, withdraw, transfer, balance, cash check, quit, help.";
44
45      /**
46       * Construct a Bank with the given name and Terminal.
47       *
48       * @param bankName the name for this Bank.
49       * @param atm  this Bank's Terminal.
50       */
51
52      public Bank( String bankName, Terminal atm )
53      {
54          this.atm      = atm;
55          this.bankName = bankName;
56          accountList   = new TreeMap();
```



```
57          month         = new Month();
58      }
59
60      /**
61       * Simulates interaction with a Bank.
62       * Presents the user with an interactive loop, prompting for
63       * banker transactions and in case of the banker transaction
64       * "customer", an account id and further customer
65       * transactions.
66       */
67
68      public void visit()
69      {
70          instructUser();
71
72          String command;
73          while (!(command =
74                   atm.readWord("banker command: ")).equals("exit")) {
75
76              if (command.startsWith("h")) {
77                  help( BANKER_COMMANDS );
78              }
79              else if (command.startsWith("o")) {
80                  openNewAccount();
81              }
82              else if (command.startsWith("r")) {
83                  report();
84              }
85              else if (command.startsWith( "c" ) ) {
86                  BankAccount acct = whichAccount();
87                  if ( acct != null )
88                      processTransactionsForAccount( acct );
89              }
90              else {
91                  // Unrecognized Request
92                  atm.println( "unknown command: " + command );
93              }
94          }
95          report();
96          atm.println( "Goodbye from " + bankName );
97      }
98
99
100     // Open a new bank account,
101     // prompting the user for information.
102
103     private void openNewAccount()
104     {
105         String accountName = atm.readWord( "Account name: " );
106         char accountType =
107             atm.readChar( "Checking/Fee/Regular? (c/f/r): " );
108         int startup = atm.readInt( "Initial deposit: " );
109         BankAccount newAccount;
110         switch( accountType ) {
111         case 'c':
112             newAccount = new CheckingAccount( startup, this );
```

```
113             break;
114         case 'f':
115             newAccount = new FeeAccount( startup, this );
116             break;
117         case 'r':
118             newAccount = new RegularAccount( startup, this );
119             break;
120         default:
121             atm.println("invalid account type: " + accountType);
122             return;
123         }
124         accountList.put( accountName, newAccount );
125         atm.println( "opened new account " + accountName
126                         + " with $" + startup );
127     }
128
129     // Prompt the customer for transaction to process.
130     // Then send an appropriate message to the account.
131
132     private void processTransactionsForAccount( BankAccount acct )
133     {
134         help( CUSTOMER_TRANSACTIONS );
135
136         String transaction;
137         while (!(transaction =
138                  atm.readWord("    transaction: ")).equals("quit")) {
139
140             if ( transaction.startsWith( "h" ) ) {
141                 help( CUSTOMER_TRANSACTIONS );
142             }
143             else if ( transaction.startsWith( "d" ) ) {
144                 int amount = atm.readInt( "    amount: " );
145                 atm.println("    deposited " + acct.deposit( amount ));
146             }
147             else if ( transaction.startsWith( "w" ) ) {
148                 int amount = atm.readInt( "    amount: " );
149                 atm.println("    withdrew " + acct.withdraw( amount ));
150             }
151             else if ( transaction.startsWith( "c" ) ) {
152                 int amount = atm.readInt( "    amount of check: " );
153                 atm.println("    cashed check for " +
154                             ((CheckingAccount)acct).honorCheck( amount ));
155             }
156             else if (transaction.startsWith("t")) {
157                 atm.print( "    to ");
158                 BankAccount toacct = whichAccount();
159                 if (toacct != null) {
160                     int amount = atm.readInt("    amount to transfer: ");
161                     atm.println("    transfered " +
162                                 toacct.deposit(acct.withdraw(amount)));
163                 }
164             }
165             else if (transaction.startsWith("b")) {
166                 atm.println("    current balance " +
167                             acct.requestBalance());
168             }
```



```
169             else {
170                 atm.println("    sorry, unknown transaction" );
171             }
172         }
173         atm.println();
174     }
175
176     // Prompt for an account name (or number), look it up
177     // in the account list. If it's there, return it;
178     // otherwise report an error and return null.
179
180     private BankAccount whichAccount()
181     {
182         String accountName = atm.readWord( "account name: " );
183         BankAccount account = (BankAccount) accountList.get(accountName);
184         if (account == null) {
185             atm.println("not a valid account");
186         }
187         return account;
188     }
189
190     // Action to take when a new month starts.
191     // Update the month field by sending a next message.
192     // Loop on all accounts, sending each a newMonth message.
193
194     private void newMonth()
195     {
196         month.next();
197         // for each account
198         //    account.newMonth()
199     }
200
201     // Report bank activity.
202     // For each BankAccount, print the customer id (name or number),
203     // account balance and the number of transactions.
204     // Then print Bank totals.
205
206     private void report()
207     {
208         atm.println( bankName + " report for " + month );
209         atm.println( "\nSummaries of individual accounts:" );
210         atm.println( "account  balance   transaction count" );
211         for (Iterator i = accountList.keySet().iterator();
212              i.hasNext(); ) {
213             String accountName = (String) i.next();
214             BankAccount acct = (BankAccount) accountList.get(accountName)
215             atm.println(accountName + "\t$" + acct.getBalance() + "\t\t"
216                         acct.getTransactionCount());
217         }
218         atm.println( "\nBank totals");
219         atm.println( "open accounts: " + getNumberOfAccounts() );
220         atm.println( "cash on hand: $" + getBalance());
221         atm.println( "transactions:  " + getTransactionCount());
222         atm.println();
223     }
224
```

```
225
226      // Welcome the user to the bank and instruct her on
227      //  her options.
228
229      private void instructUser()
230      {
231          atm.println( "Welcome to " + bankName );
232          atm.println( "Open some accounts and work with them." );
233          help( BANKER_COMMANDS );
234      }
235
236      // Display a help string.
237
238      private void help( String helpString )
239      {
240          atm.println( helpString );
241          atm.println();
242      }
243
244      /**
245       * Increment bank balance by given amount.
246       *
247       * @param amount the amount increment.
248       */
249
250      public void incrementBalance(int amount)
251      {
252          balance += amount;
253      }
254
255      /**
256       * Increment by one the count of transactions,
257       * for this bank.
258       */
259
260      public void countTransaction()
261      {
262          transactionCount++;
263      }
264
265      /**
266       * Get the number of transactions performed by this bank.
267       *
268       * @return number of transactions performed.
269       */
270
271      public int getTransactionCount( )
272      {
273          return transactionCount ;
274      }
275
276      /**
277       * Get the current bank balance.
278       *
279       * @return current bank balance.
280       */
```



```
281
282      public int getBalance()
283      {
284          return balance;
285      }
286
287      /**
288       * Get the current number of open accounts.
289       *
290       * @return number of open accounts.
291       */
292
293      public int getNumberOfAccounts()
294      {
295          return accountList.size();
296      }
297
298      /**
299       * Run the simulation by creating and then visiting a new Bank.
300       *
301       * <p>
302       * A -e argument causes the input to be echoed.
303       * This can be useful for executing the program against
304       * a test script, e.g.,
305       * <pre>
306       *   java Bank -e < Bank.in
307       * </pre>
308       *
309       * @param args the command line arguments:
310       *         <pre>
311       *         -e echo input.
312       *         bankName any other command line argument.
313       *         </pre>
314       */
315
316      public static void main( String[] args )
317      {
318          // parse the command line arguments for the echo
319          // flag and the name of the bank
320
321          boolean echo    = false;         // default does not echo
322          String bankName = "Faithless Trust"; // default bank name
323
324          for (int i = 0; i < args.length; i++ ) {
325              if (args[i].equals("-e")) {
326                  echo = true;
327              }
328              else {
329                  bankName = args[i];
330              }
331          }
332          Bank aBank = new Bank( bankName, new Terminal(echo) );
333          aBank.visit();
334      }
335  }
```